

# **APPENDIX A**

## **A 6502 ASSEMBLER IN BASIC**

### **INTRODUCTION**

Developing short programs for the 6502 may be done on paper, and the programs may then be entered on a 6502 board. However, if longer programs are to be developed (say more than a few dozen instructions), or else if a large number of small programs is to be developed, the convenience of an assembler becomes of significant importance. Since it is assumed that most readers seriously interested in applying a 6502 to real applications will start developing such programs, this book includes the full listing of an assembler for the 6502 written in BASIC for those who do not already have access to a 6502 assembler.

The advantage of an assembler for the 6502 written in BASIC is that it can be run on any computer equipped with BASIC which may be accessible to the user. The version of BASIC used in this program is the one available on Hewlett-Packard computers. It can be characterized as a subset of most microcomputer BASICs in that it does not include the features found on the latter ones. Using this assembler on a computer having a different BASIC will involve a translation process. However, the translation effort should be moderate, in view of the fact that most popular BASICs available on microcomputers include many more features than the one which has been used for this assembler. This assembler is therefore essentially upwardly compatible. In fact, a user who is good at programming in his BASIC will pro-

bably be capable of effecting a significant reduction in the number of instructions used for this assembler.

This assembler has been used to assemble a large number of programs for the 6502 and has performed successfully. To the best of our knowledge, it is therefore a reliable product. However, it is included here for educational purposes only and not warranted for any purpose whatsoever. A Microsoft BASIC version of this assembler will be published in the near future for readers interested in this particular version.

A complete listing of the assembler is shown in this section, and a sample output demonstrating its operation is shown below.

All the programs at the end of Chapter Four have been assembled with this assembler.

## GENERAL DESCRIPTION

ASM 65 is a complete 6502 mnemonic assembler. It recognizes all industry standard mnemonics, and will produce the standard hexadecimal listings, as shown on the example of Fig A-1.

In addition, this assembler provides the industry standard directives, with only exception the use of "." to indicate current location assignments and references. The directives available are: .BYT, .WORD, .DBYT, .TEXT. The user is referred to any manufacturer's assembler description for the details of these directives.

## USING THE ASSEMBLER

The ASM 65 is written in Hewlett-Packard 2000 series F BASIC. A description of the features of this particular BASIC implementation appear later in this section. Few changes should be needed to adapt this interpreter to other versions of BASIC to which the reader would have access.

ASM 65 operates on serial files. A minimum of three files are equipped and four are normally used. They are: the source file, the symbol table file, a temporary file, and optionally a destination file distinct from the source file.

The input file contains the assembly language instructions. It must

```

Z CAT SRC
;MEMORY BLOCK MOVE PROGRAM
;MOVES UP TO 255 BYTES FROM A TABLE STARTING AT
;LOC1 TO A TABLE STARTING AT LOC2. LENGTH OF THE
;SECTION TO BE MOVED IS IN MOVLEN.
MOVLEN  =%00
LOC1    =%200
LOC2    =%300
;
      LOX MOVLEN ;LOAD LENGTH OF MOVE TO INDEX
LOOP   LOA LOC1,X ;LOAD BYTE TO BE MOVED
      STA LOC2,X ;STORE BYTE TO BE MOVED
      DEX ;COUNT DOWN
      BPL LOOP ;IF NOT DONE, MOVE NEXT BYTE
      RTS ;DONE

Z RUN ASM65
SOURCE FILE ?SRC
OBJECT FILE ?DEST
PRINTOUT ?YES
ASSEMBLY BEGINS...

      ;MEMORY BLOCK MOVE PROGRAM
      ;MOVES UP TO 255 BYTES FROM A TABLE STARTING AT
      ;LOC1 TO A TABLE STARTING AT LOC2. LENGTH OF THE
      ;SECTION TO BE MOVED IS IN MOVLEN.
      MOVLEN  =%00
      LOC1    =%200
      LOC2    =%300
      ;
0000: A6 00          LOX MOVLEN      ;LOAD LENGTH OF MOVE TO INDEX
0002: B0 00 02      LOOP   LOA LOC1,X ;LOAD BYTE TO BE MOVED
0005: 9D 00 03          STA LOC2,X  ;STORE BYTE TO BE MOVED
0008: CA            DEX             ;COUNT DOWN
0009: 10 F7          BPL LOOP      ;IF NOT DONE, MOVE NEXT BYTE
000B: 60            RTS             ;DONE

SYMBOL TABLE:
MOVLEN      0000      LOC1      0200      LOC2      0300
LOOP        0002
DONE

```

**Fig A1: Using the ASM 65 Assembler**

therefore contain ASCII text, and must be structured as per the rules of the assembler syntax (described in the next section). In general, the input lines can be written in free format, with the fields separated by one or more spaces. However, any label must start in column one. Any line without a label may not start in column one.

The assembler will automatically format the comment field on the output file. However it will not format the other fields within the instructions so that the user may tabulate his input statements in any reasonable way for clarity. This feature is intended to improve readability.

The output file is also ASCII text, including the representation of all numbers. The output file may optionally be printed after the second pass of the assembler has been executed. A prompt is printed on the listing, or appears on the screen as "PRINTOUT?" and the user may specify "yes" or "no."

The assembler provides extensive diagnostics and will describe all errors it has identified, then list them on the output.

In this implementation, the error printout may contain various field markers such as operator field limiters ("!"), and the internal unresolved reference delimiter ("\*\*\*").

The symbol table gives the usual hexadecimal representation for all symbolic labels used by the program. An example is shown in Fig A-2.

SYMBOL TABLE;					
MOVLEN	0000	LOC1	0200	LOC2	0300
LOOP	0002				
DONE					

**Fig A-2: The Symbol Table**

## SYNTAX

### Constants

Constants may be expressed in any of the four usual number representations:

- Hexadecimal: the constant must be preceded by a "\$". Example: "LDA \$20" will load the accumulator from memory address "20" hexadecimal.
- Binary: it must be preceded by a "%". Example: "LDA %11111111" will load the accumulator with all ones.
- Decimal: usual representation. Example "LDA #0" will load the accumulator with the decimal value zero.
- ASCII: must be preceded by a "'". Example: "LDA 'A'" will load the ASCII code for A into the accumulator.

### Arithmetic Expressions

Arithmetic expressions may be used in the operator field, in a label

assignment, or in a memory allocation instruction.

The operand in an arithmetic expression may be a number expressed in any representation, or a label, or a "." (the current location symbol) or any combination of those. The legal operators are "+" and "-". In the case where more than one operator is used, the arithmetic expression will be evaluated from left to right.

## Comments

Comments must be preceded by a ";". They may begin in any column including column one. All comments will be justified in the middle of the output sheet unless they begin in column one.

## Memory Assignments

Memory assignments are performed by one or more of the four directives:

- .BYT           – Assigns one byte of data to one memory location.
- .WORD         – Assigns two bytes of data to two consecutive memory locations, low order byte first.
- .DBYT         – Assigns two bytes of data to two consecutive memory locations, high order byte first.
- .TEXT         – Converts an ASCII string to hex data, and stores it in consecutive memory locations. The string must be delimited by two identical non-blank characters.

There is no end directive – an end-of-file is used instead.

Example of a memory assignment:

```
.BYT    $2A, WORDCONST
.WORD    2, %10
```

## HP2000F BASIC:

Hewlett-Packard BASIC is different from many common mini- and microcomputer BASICs, but is easily adapted. The following is a list

of features which differ from most BASICs, or from the Dartmouth standard.

## Files

Files are declared in a FILES statement at the beginning of the program and are numbered in the order in which they appear in it. The ASSIGN statement assigns a file specified by its first argument to a file number specified by the second argument. The third argument is a dummy variable. A star appearing in a FILES statement means a file will later be assigned to that file position by an ASSIGN statement. The READ statement reads the file. Its first argument, preceded by a "#", is the file number of the file to be read from. If the record number is one, and there is no semicolon, the statement serves to reset the file pointer to 0, as in "READ #2, 1". Any arguments after the semicolon are those variables to be read.

The PRINT statement is similar to the read statement. It also has a special form, "PRINT #2,END", which makes an end-of-file marker on the file.

The IF END # THEN statement operates in a way analogous to a vectored interrupt. When an end-of-file occurs on a read, program execution will continue at the line number mentioned after the THEN, instead of causing the program to crash. This will occur even if the computer is not currently executing the statement: i.e., the end-of-file vector need only be specified once, unless it needs to be changed.

## Strings

Strings are one dimensional, and can only be dimensioned as such. To assign 0 (zero) length to a string, or clear it, a statement of the type "L\$=" " is used. Characters in a string are referenced as follows: to reference a substring within a larger string, the form "T\$(a,b)" is used where a and b are expressions signifying respectively the first and last character addresses in the main string of the desired substring. Characters in a string are addressed from left to right, starting at 1. Example: if A\$ = "ABCDE" and the statement "B\$ = A\$(2,3)" is executed, B\$ will become "BC".

The form "T\$(a)" references all characters in T\$ starting with character #a and continuing on to the end of T\$.

Example: if A\$ = "12345", A\$(3) means the substring "345".

The string functions CHR\$ and ASC\$, which respectively convert an ASCII decimal number into a one-character string, and a one-character string into its decimal ASCII equivalent are not available, so ASM65 reads a string of ordered ASCII characters from a system file called \$ASCIIF, which it then uses for number and string conversion.

MAX returns the maximum of 2 values.

Example: "B=11 MAX 9" would yield 11.

MIN returns the minimum of 2 values.

LIN when in a print statement adds amount of linefeed specified in its argument to output.

The above definitions are intended only as guidelines for the translation of ASM65 into other versions of BASIC.

**Fig A-3: 6502 Assembler Listing**  
copyright © 1979, Sybex Inc.

ASM65

```

10 REM : ***** 6502 MNEMONIC ASSEMBLER, VERSION 2.0 *****
20 REM
30 REM : WRITTEN IN HP2000F TSS BASIC.
40 REM : CAN BE USED WITH ALL 65XX PROCESSORS AS MADE BY COMMODORE,
50 REM : SYNERTEK, AND ROCKWELL.
60 REM : ALL MNEMONICS AND DIRECTIVES ARE INDUSTRY STANDARD, WITH
70 REM : THE EXCEPTION OF THE USE OF '.' FOR CURRENT ADDRESS.
80 R=10
90 T9=0
100 A=0
110 DIM L$(72),M$(72),O$(72),C$(72),Z$(72),P$(72),T$(72)
120 DIM A$(72),N$(72)
130 DIM I$(72)
140 L=0
150 FILES *,SYMTAB,TEMP,*,*,$ASCIIF
160 PRINT "SOURCE FILE ";
170 INPUT T$
180 PRINT "OBJECT FILE ";
190 INPUT O$
200 ASSIGN T$,1,08
210 ASSIGN O$,4,08
220 READ #1,1
230 PRINT #2,1
240 PRINT #3,1
250 RB=0
260 PRINT "PRINTOUT ";
270 INPUT I$
280 IF I$ <> "NO" THEN 300
290 RB=1
300 PRINT "ASSEMBLY BEGINS..."
310 C=0

```

```

320 IF END #1 THEN 2440
330 L$=""
340 I$=""
350 M$=""
360 O$=""
370 C$=""
380 Z$=""
390 L=L+1
400 REM***** SEPARATE TOKENS, STORE LABEL ASSIGNMENTS *****
410 READ #1;I$
420 T5=C
430 IF I$="" THEN 830
440 P=1
450 P$=""
460 GOSUB 3970
470 IF P1=0 THEN 510
480 IF P1=1 THEN 800
490 C$=I$(P1)
500 I$=I$(1,P1-1)
510 IF I$(1,1)="" THEN 590
520 GOSUB 3790
530 L$=P$
540 IF L$ <> "." THEN 590
550 M$="."
560 GOSUB 4940
570 L$=""
580 GOTO 840
590 GOSUB 3790
600 M$=P$
610 IF M$(1,3)=".WO" THEN 3110
620 IF M$(1,3)=".TE" THEN 3110
630 IF M$(1,3)=".BY" THEN 3110
640 IF M$(1,3)=".DB" THEN 3110
650 IF M$ <> "" THEN 850
660 C$=C$(1,34)
670 IF LEN(L$) <> 0 THEN 700
680 I$=I$(1,19)
690 GOTO 820
700 GOSUB 3790
710 M$=P$
720 IF LEN(M$) <> 0 THEN 750
730 T1=C
740 GOTO 780
750 GOSUB 4070
760 IF T4=2 THEN 830
770 T1=F1
780 PRINT #2;L$,T1
790 PRINT #2; END
800 I$=I$(1,LEN(I$) MIN 55)
810 Z$(17,17+LEN(I$))=I$
820 Z$(LEN(I$)+19 MAX 38) MIN 72)=C$
830 PRINT #3;Z$,T5
840 GOTO 320
850 IF M$(1,1) <> "." THEN 1050
860 P$=""
870 GOSUB 3970
880 IF P1>0 THEN 910
890 PRINT "MISSING '=' IN LINE #1L
900 GOTO 3090
910 P=P1+1
920 GOSUB 3790
930 IF P$(1,1) <> "" THEN 960
940 PRINT "MISSING ARGUMENT IN LINE #1L
950 GOTO 3090
960 M$=P$

```



```

970 GOSUB 4070
980 IF T4 <> 2 THEN 1010
990 PRINT "ILLEGAL FORWARD REFERENCE IN LINE " #L
1000 GOTO 3090
1010 T1=C
1020 C=F1
1030 IF L$ <> "" THEN 780
1040 GOTO 800
1050 RESTORE 5710
1060 IF M$="" THEN 1140
1070 FOR I=1 TO 56
1080 READ T$
1090 IF T$=M$ THEN 1130
1100 NEXT I
1110 PRINT "UNKNOWN OPCODE IN LINE " #L
1120 GOTO 3090
1130 O=I
1140 IF L$="" THEN 1170
1150 PRINT #2;L$,C
1160 PRINT #2; END
1170 GOSUB 3750
1180 O$=P$
1190 I$=(P-LEN(O$)-1,P-LEN(O$)-1)="!"
1200 REM***** FIND ADDRESSING MODES, LOAD EFFECTIVE ADDRESS *****
1210 IF O$ <> "" THEN 1240
1220 M=1
1230 GOTO 2200
1240 IF O$ <> "A" THEN 1270
1250 M=2
1260 GOTO 2200
1270 IF O$[1,1] <> "*" THEN 1320
1280 M=3
1290 P=P+1
1300 M$=O$[2]
1310 GOTO 1870
1320 IF M$[1,1] <> "B" THEN 1460
1330 IF M$="BIT" THEN 1460
1340 M=12
1350 M$=O$
1360 GOSUB 4070
1370 IF T4 <> 2 THEN 1400
1380 A=-200
1390 GOTO 1970
1400 A=F1-C-2
1410 IF A >= 0 THEN 1430
1420 A=256+A
1430 IF ABS(F1-C) <= 127 THEN 1970
1440 PRINT "BRANCH OUT OF RANGE IN LINE " #L
1450 GOTO 3090
1460 P$="( "
1470 P=P-LEN(O$)
1480 GOSUB 3970
1490 P5=P1
1500 P$=","
1510 GOSUB 3970
1520 P6=P1
1530 P7=0
1540 IF NOT P6 THEN 1610
1550 IF I$[P6+1,P6+1] <> "X" THEN 1580
1560 P7=I
1570 OOTQ 1610
1580 IF I$[P6+1,P6+1]="Y" THEN 1610
1590 PRINT "BAD ADDRESSING MODE IN LINE " #L
1600 GOTO 3090
1610 IF P5 <> 0 THEN 1780

```

```

1620 GOSUB 3790
1630 N%=P%
1640 IF NOT P6 OR NOT P7 THEN 1670
1650 M=5
1660 GOTO 1710
1670 IF NOT P6 THEN 1700
1680 M=6
1690 GOTO 1710
1700 M=4
1710 GOSUB 4070
1720 A=F1
1730 IF T4 <> 2 THEN 1750
1740 A=-1000
1750 IF ABS(A) <= 255 THEN 1970
1760 M=M+3
1770 GOTO 1970
1780 GOSUB 3790
1790 N%=P%[2]
1800 IF NOT P6 OR NOT P7 THEN 1830
1810 M=10
1820 GOTO 1870
1830 IF NOT P6 THEN 1860
1840 M=11
1850 GOTO 1870
1860 M=13
1870 GOSUB 4070
1880 A=F1
1890 IF (M <> 10 AND M <> 11) OR A <= 255 THEN 1920
1900 PRINT "VALUE TOO LARGE FOR ZERO PAGE IN LINE *IL
1910 GOTO 3090
1920 IF T4 <> 2 THEN 1970
1930 A=-1000
1940 IF M=13 THEN 1970
1950 A=-200
1960 REM***** PRINT OPCODES & EA ON FILE *****
1970 IF A >= 0 THEN 2070
1980 Z%[10,11]="**"
1990 C=C+1
2000 IF M <> 12 THEN 2020
2010 Z%[11,11]="R"
2020 W9=A+256
2030 IF W9 >= 0 THEN 2200
2040 Z%[13,14]="**"
2050 C=C+1
2060 GOTO 2200
2070 R=16
2080 I=A
2090 GOSUB 4940
2100 T%=A%
2110 A%="000"
2120 A%[4]=T%
2130 IF (M >= 3 AND M <= 6) OR (M >= 10 AND M <= 12) THEN 2180
2140 Z%[13,14]=A%[LEN(A%)-3,LEN(A%)-2]
2150 Z%[10,11]=A%[LEN(A%)-1]
2160 C=C+2
2170 GOTO 2200
2180 Z%[10,11]=A%[LEN(A%)-1]
2190 C=C+1
2200 R=16
2210 I=T%
2220 GOSUB 4940
2230 T%="000"
2240 T%[4]=A%
2250 Z%[1,4]=T%[LEN(T%)-3]
2260 RESTORE 5140

```

```

2270 FOR I=1 TO (O-1)*13+M
2280 READ T$
2290 NEXT I
2300 IF T$ <> " " THEN 2370
2310 IF M>6 OR M<4 THEN 2350
2320 M=M+3
2330 C=T$
2340 GOTO 1970
2350 PRINT 'ILLEGAL ADDRESSING MODE IN LINE 'I;L
2360 GOTO 3090
2370 Z$(7,8)=T$
2380 Z$(5,5)=" "
2390 C=C+1
2400 Z$(17,17+LEN(I$))=I$
2410 Z$[(19+LEN(I$)) MAX 38]=C$(1,72-((19+LEN(I$)) MAX 38))
2420 PRINT #3;Z$;T$
2430 GOTO 320
2440 REM***** SECOND PASS: RESOLVE FWD REFERENCES *****
2450 PRINT #2; END
2460 PRINT #3; END
2470 READ #2;I
2480 L=0
2490 READ #3;I
2500 PRINT #4;I
2510 IF ENO #3 THEN 2870
2520 P=1
2530 READ #3;I$,T$
2540 L=L+1
2550 IF I$="" THEN 2850
2560 P$=""
2570 GOSUB 3970
2580 IF P1=0 OR P1=17 THEN 2610
2590 P=P1
2600 I$(P,P)=I$
2610 IF I$(10,10) <> "*" THEN 2850
2620 GOSUB 3790
2630 N$=P$
2640 IF N$(1,1) <> "(" THEN 2660
2650 N$=N$(2)
2660 GOSUB 4070
2670 IF T4 <> 2 THEN 2700
2680 PRINT 'IRRESOLVABLE FWD REF / BAD LABEL IN LINE 'I;L
2690 GOTO 3090
2700 I=F1
2710 IF I$(11,11) <> "R" THEN 2750
2720 I=F1-T5-2
2730 IF I >= 0 THEN 2750
2740 I=I+254
2750 R=16
2760 GOSUB 4940
2770 T$=A$
2780 A$="000"
2790 A$(4)=T$
2800 IF I$(13,14) <> "***" THEN 2840
2810 I$(13,14)=A$(LEN(A$)-3,LEN(A$)-1)
2820 I$(10,11)=A$(LEN(A$)-1)
2830 GOTO 2850
2840 I$(10,11)=A$(LEN(A$)-1)
2850 PRINT #4;I$
2860 GOTO 2510
2870 PRINT #4; END
2880 IF R8=1 THEN 3080
2890 IF END #4 THEN 2940
2900 READ #4;I
2910 READ #4;I$

```

```

2920 PRINT I$
2930 GOTO 2910
2940 READ #2,1
2950 PRINT LIN(2);"SYMBOL TABLE:"
2960 IF END #2 THEN 3080
2970 FOR I6=1 TO 3
2980 READ #2;Q$,T$
2990 R=16
3000 I=T$
3010 GOSUB 4940
3020 T$="0000"
3030 T$[LEN(T$)+1]=A$
3040 PRINT TAB((I6-1)*25+1);Q$;TAB((I6-1)*25+13);T$[LEN(T$)-3];
3050 NEXT I6
3060 PRINT
3070 GOTO 2970
3080 END
3090 PRINT "<"I$">"
3100 END
3110 REM$***** PROCESS MEMORY LOADS *****$
3120 Q7=1
3130 IF M$(2,3) <> "TE" THEN 3260
3140 IF Q7 <> 1 THEN 3190
3150 GOSUB 3750
3160 P=P-LEN(P$)
3170 Q$=I$(P,P)
3180 P=P+1
3190 IF P <= 72 THEN 3220
3200 PRINT "BAD DELIMITER IN LINE "I$
3210 GOTO 3090
3220 P$(1)=""
3230 P$(2,2)=I$(P,P)
3240 IF P$(2,2)=Q$ THEN 320
3250 GOTO 3280
3260 GOSUB 3790
3270 Z$=""
3280 P=P+1
3290 IF LEN(P$)=0 THEN 320
3300 N$=P$
3310 GOSUB 4070
3320 IF T4 <> 2 THEN 3350
3330 PRINT "BAD LABEL IN MEMORY ASSIGNMENT OF LINE "I$
3340 GOTO 3090
3350 R=16
3360 I=F1
3370 GOSUB 4940
3380 T$=A$
3390 A$="000"
3400 A$[4]=T$
3410 IF M$(2,2) <> "W" THEN 3460
3420 Z$(10,11)=A$[LEN(A$)-3,LEN(A$)-2]
3430 Z$(7,8)=A$[LEN(A$)-1]
3440 C=C+2
3450 GOTO 3560
3460 IF M$(2,2)="D" THEN 3530
3470 IF F1<256 THEN 3500
3480 PRINT "NUMBER TOO LARGE IN MEMORY ASSIGNMENT OF LINE "I$
3490 GOTO 3090
3500 Z$(7,8)=A$[LEN(A$)-1]
3510 C=C+1
3520 GOTO 3560
3530 Z$(7,8)=A$[LEN(A$)-3,LEN(A$)-2]
3540 Z$(10,11)=A$[LEN(A$)-1]
3550 C=C+1
3560 I=T$

```

```

3570 R=16
3580 GOSUB 4940
3590 T9="000"
3600 T9[4]=A$
3610 Z$[1,4]=T9[LEN(T9)-3]
3620 Z$[5,5]=":"
3630 IF 07 <> 1 THEN 3700
3640 IF LEN(L$)=0 THEN 3670
3650 PRINT #2;L$,T5
3660 PRINT #2; END
3670 Z$[17,17+LEN(I$)]=I$
3680 Z$[(19+LEN(I$)) MAX 38]=C$[1,72-(19+LEN(I$)) MAX 38]
3690 GOTO 3710
3700 Z$=Z$[1,15]
3710 Q7=0
3720 PRINT #3;Z$,T5
3730 T5=C
3740 GOTO 3130
3750 REM ***** ROUTINE TO ISOLATE TOKEN *****
3760 REM : STARTS LOOKING FOR TOKEN AT P, PUTS IT IN P$, AND
3770 REM : UPDATES P. IF ENTERED HERE, STOPS SCAN AT ' '.
3780 T9=1
3790 REM : IF ENTERED HERE, STOPS SCAN AT ' ', ',', ')', '='.
3800 FOR I1=P TO LEN(I$)
3810 IF I$[I1,I1] <> " " THEN 3830
3820 NEXT I1
3830 P$=""
3840 FOR I2=I1 TO LEN(I$)
3850 IF I$[I2,I2]=" " THEN 3920
3860 IF T9=1 THEN 3900
3870 IF I$[I2,I2]="," THEN 3920
3880 IF I$[I2,I2]=")" THEN 3920
3890 IF I$[I2,I2]="=" THEN 3920
3900 P$[LEN(P$)+1]=I$[I2,I2]
3910 NEXT I2
3920 P=I2
3930 IF LEN(P$) <> 0 THEN 3950
3940 P=P+1
3950 T9=0
3960 RETURN
3970 REM ***** FIND SYMBOL ROUTINE *****
3980 REM : RETURNS P1=SYMLOC IF IT IS FOUND, P1=0
3990 REM : IF SYMBOL NOT FOUND
4000 FOR I=P TO LEN(I$)
4010 IF I$[I,I]=P$[1,1] THEN 4050
4020 NEXT I
4030 P1=0
4040 RETURN
4050 P1=I
4060 RETURN
4070 REM ***** NUMERIC STRING INTERPRETER *****
4080 REM : SIMPLIFIES STRINGS OF LABELS AND NUMERIC EXPRESSIONS
4090 REM : OF NUMBERS IN ANY BASE, PLUS ASCII CONSTANTS.
4100 F1=W=0
4110 A$=""
4120 FOR I=1 TO LEN(N$)
4130 IF N$[I,I]="+" THEN 4180
4140 IF N$[I,I]="-" THEN 4180
4150 IF N$[I,I]="*" THEN 4610
4160 A$[LEN(A$)+1]=N$[I,I]
4170 NEXT I
4180 IF A$ <> "." THEN 4210
4190 F2=C
4200 GOTO 4480
4210 IF A$[1,1]>"Z" THEN 4350
4220 IF A$[1,1]<"A" THEN 4350

```

```

4230 READ #2,I
4240 IF END #2 THEN 4330
4250 READ #2;T$,T1
4260 IF T$ <> A$ THEN 4240
4270 F2=T1
4280 T4=3
4290 IF END #2 THEN 4320
4300 READ #2;T$,T1
4310 GOTO 4300
4320 GOTO 4480
4330 T4=2
4340 RETURN
4350 IF A$(1,1) <> "" THEN 4390
4360 A$=A$(2)
4370 GOSUB 4640
4380 GOTO 4480
4390 B=10
4400 IF A$(1,1) <> "X" THEN 4430
4410 B=2
4420 GOTO 4450
4430 IF A$(1,1) <> "*" THEN 4460
4440 B=16
4450 A$=A$(2)
4460 GOSUB 4750
4470 F2=F
4480 IF W=2 THEN 4510
4490 F1=F1+F2
4500 GOTO 4520
4510 F1=F1-F2
4520 IF I >= LEN(N$) THEN 4610
4530 T$="+-"
4540 FOR W=1 TO LEN(T$)
4550 IF T$(W,W)=N$(I,I) THEN 4590
4560 NEXT W
4570 PRINT "ILLEGAL OPERATOR IN LINE ";L
4580 GOTO 3090
4590 A$=""
4600 GOTO 4170
4610 T4=0
4620 RETURN
4630 REM ***** ASCII CHARACTER TO NUMBER CONVERTER *****
4640 A$=A$(1,1)
4650 F2=0
4660 READ #5,I
4670 READ #5;T$
4680 FOR I=1 TO 72
4690 IF A$(1,1)=T$(I,I) THEN 4740
4700 F2=F2+1
4710 NEXT I
4720 F2=F2-8
4730 GOTO 4670
4740 RETURN
4750 REM ***** MULTI-RADIX STRING TO NUMBER CONVERTER *****
4760 REM : B IS BASE OF NUMBER IN A$, F IS PRODUCT.
4770 F=0
4780 I1=0
4790 FOR I2=LEN(A$) TO 1 STEP -1
4800 RESTORE 4910
4810 FOR N=0 TO B-1
4820 READ F$
4830 IF F$=A$(I2,I2) THEN 4870
4840 NEXT N
4850 PRINT "BAD NUMBER IN LINE ";L
4860 GOTO 3090
4870 F=F+N*B^I1

```

```

4880 I1=I1+1
4890 NEXT I2
4900 RETURN
4910 DATA "0","1","2","3","4","5","6","7","8","9","A","B","C","D"
4920 DATA "E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S"
4930 DATA "T","U","V","W","X","Y","Z"
4940 REM ***** MULTI-RADIX NUMBER TO STRING CONVERTER
4950 REM : I IS INPUT NUMBER, R IS BASE THAT A$ WILL BE AS PRODUCT.
4960 A$=""
4970 T=I
4980 FOR N=20 TO 0 STEP -1
4990 IF T/R^N >= 1 THEN 5020
5000 NEXT N
5010 N=N-1
5020 Q=INT(T/R^N)
5030 IF Q <= R-1 THEN 5050
5040 Q=0
5050 T=T-Q*R^N
5060 RESTORE 4910
5070 FOR S=0 TO Q
5080 READ T$
5090 NEXT S
5100 A$=LEN(A$)+1]=T$
5110 IF N>0 THEN 5010
5120 RETURN
5130 REM ***** DPCODE TABLE *****
5140 DATA " ", "69", "65", "75", "6D", "7D", "79", "61", "71", " ", " "
5150 DATA " ", "29", "25", "35", "2D", "3D", "39", "21", "31", " ", " "
5160 DATA "0A", "06", "16", "0E", "1E", " ", " ", " ", " ", " "
5170 DATA " ", " ", " ", " ", " ", " ", " ", " ", "90", " "
5180 DATA " ", " ", " ", " ", " ", " ", " ", " ", "B0", " "
5190 DATA " ", " ", " ", " ", " ", " ", " ", " ", "F0", " "
5200 DATA " ", " ", "24", " ", "2C", " ", " ", " ", " ", " "
5210 DATA " ", " ", " ", " ", " ", " ", " ", " ", "30", " "
5220 DATA " ", " ", " ", " ", " ", " ", " ", " ", "D0", " "
5230 DATA " ", " ", " ", " ", " ", " ", " ", " ", "10", " "
5240 DATA "00", " ", " ", " ", " ", " ", " ", " ", " ", " "
5250 DATA " ", " ", " ", " ", " ", " ", " ", " ", "50", " "
5260 DATA " ", " ", " ", " ", " ", " ", " ", " ", "70", " "
5270 DATA "18", " ", " ", " ", " ", " ", " ", " ", " ", " "
5280 DATA "D8", " ", " ", " ", " ", " ", " ", " ", " ", " "
5290 DATA "58", " ", " ", " ", " ", " ", " ", " ", " ", " "
5300 DATA "B8", " ", " ", " ", " ", " ", " ", " ", " ", " "
5310 DATA " ", "C9", "C5", "D5", "CD", "DD", "D9", "C1", "D1", " ", " "
5320 DATA " ", "E0", "E4", " ", "EC", " ", " ", " ", " ", " "
5330 DATA " ", "C0", "C4", " ", "CC", " ", " ", " ", " ", " "
5340 DATA " ", " ", "C6", "D6", "CE", "DE", " ", " ", " ", " "
5350 DATA "CA", " ", " ", " ", " ", " ", " ", " ", " ", " "
5360 DATA "88", " ", " ", " ", " ", " ", " ", " ", " ", " "
5370 DATA " ", "49", "45", "55", "4D", "5D", "59", "41", "51", " ", " "
5380 DATA " ", " ", "E6", "F6", "EE", "FE", " ", " ", " ", " "
5390 DATA "E8", " ", " ", " ", " ", " ", " ", " ", " ", " "
5400 DATA "C8", " ", " ", " ", " ", " ", " ", " ", " ", " "
5410 DATA " ", " ", " ", " ", "4C", " ", " ", " ", "6C", " "
5420 DATA " ", " ", " ", " ", "20", " ", " ", " ", " ", " "
5430 DATA " ", "A9", "A5", "B5", "AB", "BD", "B9", "A1", "B1", " ", " "
5440 DATA " ", "A2", "A6", "B6", "AE", " ", "BE", " ", " ", " "
5450 DATA " ", "A0", "A4", "B4", "AC", "BC", " ", " ", " ", " "
5460 DATA "4A", " ", "46", "56", "4E", "5E", " ", " ", " ", " "
5470 DATA "EA", " ", " ", " ", " ", " ", " ", " ", " ", " "
5480 DATA " ", "09", "05", "15", "0D", "1D", "19", "01", "11", " ", " "
5490 DATA "48", " ", " ", " ", " ", " ", " ", " ", " ", " "
5500 DATA "08", " ", " ", " ", " ", " ", " ", " ", " ", " "
5510 DATA "68", " ", " ", " ", " ", " ", " ", " ", " ", " "
5520 DATA "28", " ", " ", " ", " ", " ", " ", " ", " ", " "

```

## 6502 APPLICATIONS

```

5530 DATA "2A","26","36","2E","3E","6E","7E"
5540 DATA "6A","66","76","6E","7E"
5550 DATA "40","E9","E5","F5","ED","FD","F9","E1","F1"
5560 DATA "60"
5570 DATA "E9","E5","F5","ED","FD","F9","E1","F1"
5580 DATA "38"
5590 DATA "F8"
5600 DATA "78"
5610 DATA "85","95","8D","9D","99","81","91"
5620 DATA "86","96","8E"
5630 DATA "84","94","8C"
5640 DATA "AA"
5650 DATA "AB"
5660 DATA "BA"
5670 DATA "8A"
5680 DATA "9A"
5690 DATA "98"
5700 REM ***** MNEMONIC TABLE *****
5710 DATA "ADC","AND","ASL","RCC","RCS","REQ","RIT","RMI","RNE"
5720 DATA "RPL","RRK","RVC","RVS","CLC","CLD","CLI","CLV","CMP","CFX","CFY"
5730 DATA "DEC","DEX","DEY","EOR","INC","INX","INY","JMP","JSR","LDA","LDX"
5740 DATA "LDY","LSR","NOP","ORA","FHA","PHP","PLA","PLP","ROL","ROR","RII"
5750 DATA "RTS","SRC","SEC","SED","SEI","STA","STX","STY","TAX","TAY","TSX"
5760 DATA "TXA","TXS","TYA"
5770 END

```